Adaptive Image Analysis for Aerial Surveillance

Paul Robertson and J. Michael Brady, Robotics Research Group, Oxford University

THE AUTHORS DESCRIBE A COMPUTER-VISION SYSTEM THAT WOULD AUTOMATICALLY MONITOR ITS OWN PERFORMANCE AND DYNAMICALLY ADAPT TO CHANGING SITUATIONS AND REQUIREMENTS. THEIR APPROACH IS BASED ON COMPUTATIONAL REFLECTION AND CONTROL-SYSTEM THEORY.

IN MILITARY INTELLIGENCE APPLICATIONS, robustness and predictability are crucial because the fate of a battle and those who are fighting it is at stake. Computer-vision systems must have robust, predictable performance before they will be fully embraced in military applications and other safety-critical applications such as medical imaging and intelligent transportation systems.

We try to deal with these requirements by keeping systems simple. Experience has shown that the systems that work best are those whose complexity has been minimized and that have been manually tweaked until they perform reasonably well as long as nothing changes too much.

Unfortunately, things do change. Sometimes, what changes is the environment, such as the lighting, altitude, or weather. Other times, it is the technology, such as a new camera or the replacement of an algorithm. Even small changes often cause systems to break unpredictably.

We can address such fragility in several ways. One approach is to build in more and more application-specific knowledge. However, this can lead to unmanageable complexity and performance characteristics that are not understandable.

The dominant approach has been to develop mathematical analysis of imagery and applications. This has led to some highly encouraging advances, but the real world is extremely complex. The performance of computer vision systems falls far short of human or animal perception. In short, mathematical analysis is necessary but unlikely to ever be sufficient.

We've developed an alternative approach: build architectures that support the integration of knowledge within a framework that is both simple enough and retains enough semantic information to permit analysis of its behavior. This approach has three advantages:

- We can make semantics explicit and then construct the interactions algorithmically to satisfy certain constraints. This way, the semantics of the parts are known. Although the specific pattern of interactions might not be understood, the constraints that the algorithm satisfied in composing them can be understood.
- This approach can provide graceful aggregation and degradation of competence. When we can add (or use) better image filters, we expect incrementally better results; when we can't use certain filters because the image conditions don't permit it, we expect slightly lower performance because we rely on poorer filters. The process of building a system that can make these transitions smoothly is difficult and haphazard when it depends on manually constructed ad hoc programming.
- We can incorporate the previous two advantages into a manageable framework. Producing good image-analysis results is difficult. We always want to use the best filters possible and to mobilize as much knowledge into the interpretation as possible.

1094-7167/99/$10.00 © 1999 IEEE

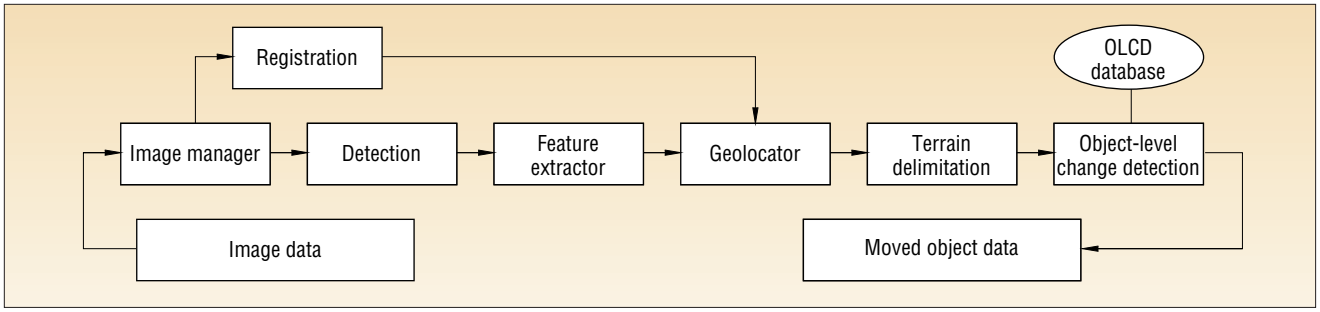

Figure 1. A typical pipelined architecture for aerial surveillance.

Creating stable, adaptive image-analysis systems

Image-processing programs are typically constructed from a number of filters whose results are combined to produce good results. Image interpretation is divided into a number of processing stages, so filters are serially connected.

Figure 1 shows a typical pipelined architecture for a military aerial-surveillance program. The architecture is a network of interconnected filters. The raw image is the input signal to this system. Successive stages of the program transform this signal. These networks, which are manually pieced together, tend to be simple because the systems get harder to manage as complexity grows. No theory exists for what conditions must be satisfied to make the resulting system stable or for predicting how changes in components will affect overall system performance.

Introducing adaptability into this system could increase its robustness. However, this might decrease the system's stability. For the system to self adapt, it must meet these requirements.

- It must be able to reason about its computational intent and state to assemble the filters in a way that is consistent with the system's task. This sort of self-knowledge is usually not required by systems that are assembled by human programmers and that remain static throughout their lifetime. *Reflection* (which we discuss in the next section) provides an approach to managing this kind of self-information.
- Its structure must allow filters to be added and removed without complicated entanglements with other software. The solution, therefore, requires a clean architectural approach.
- Human-assembled software is carefully tested in a controlled environment before it is used in a live situation. Software that restructures itself dynamically needs reassurance that the restructured software works as expected. One approach to this is to generate provably correct code. Another way is to make the code self-monitoring and to have an architecture that is simple enough to facilitate reasoning about the overall system's runtime state. One such architecture structures the software update process as a control system.

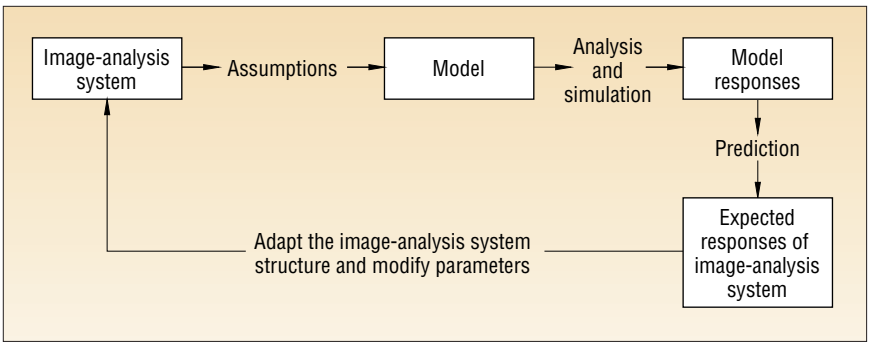

Figure 2. The design process of an image-analysis system.

We can summarize these requirements with the slogan "Self-adaptation equals reflection plus monitoring plus control." Monitoring lets the program measure deviations from an intended state; control provides a way of restructuring the program to move towards the intended state; and reflection provides the introspection capability necessary to support monitoring and the access to embedded semantics necessary to facilitate changes in the program.

Reflection

Brian Smith introduced the term *procedural reflection* in 1982 to describe the use of semantic reification to help write programs that can manipulate, to some degree, their own computational nature.[1] Reflection opens up the computational state so that it can be manipulated within the language or system. In addition to the basic program capabilities, support for reflection comes from these features:

- *An embedded semantic account*, which is the reified semantics.
- *A causal connection* between the embedded semantic account and the program's semantics. This allows interrogation (introspection) of the program's computation state; by modifying the data structures or functions, we can modify the program's semantics.
- *An appropriate level of detail*. Typically, the entire state of the computation doesn't need to be interrogated, and the entire semantics doesn't need to be modified. In solving real problems, a sense of proportion is necessary, and success is achieved by selecting which subset of the semantics to reify. This selection requires an analysis of the reflective system's goals.

Reflection lets software components modify their own semantics. By reifying the semantics of the image-analysis system design process (see Figure 2), the individual filters can participate in the program's adaptation.

Building an adaptive architecture

We have chosen the problem of image segmentation to focus our efforts on developing and demonstrating the benefits of an adaptive computing approach to aerial image analysis. Image-segmentation partitions an

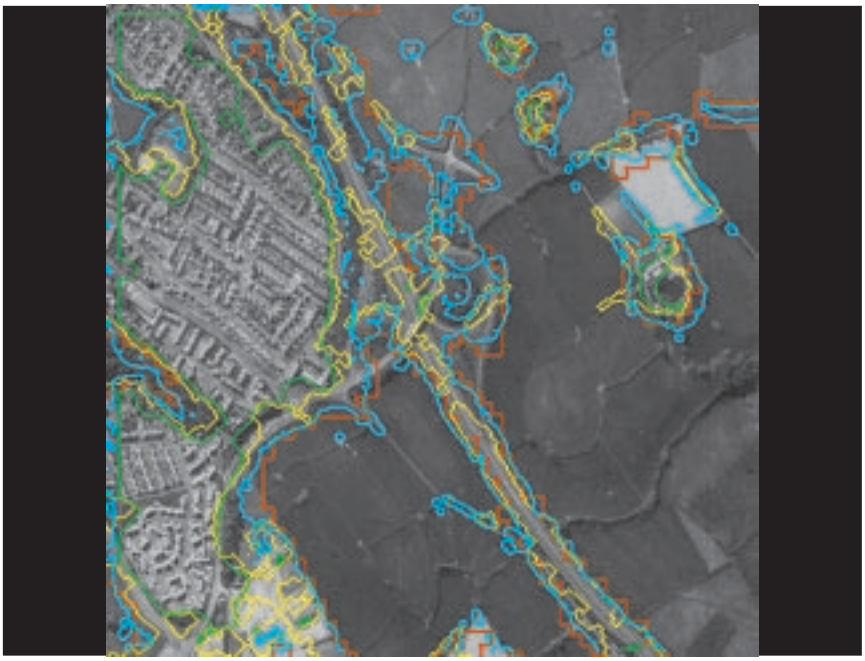

Figure 3. Combining evidence from multiple filters. red = belief network; green = fractile estimate; yellow = wavelet estimate; blue = combined feature images from belief network, fractal, and wavelet. (Image provided by Paul Ducksbury, Defense Evaluation Research Agency, Malvern, UK.)

image into disjoint regions, each of which is homogeneous in some property. Examples of such properties include

- A threshold value or range such that all pixels in the region have brightness values in that range—for example, (23, 47).
- Measures of texture, such as co-occurrences, fractal measures, or wavelet local energy.[2]
- Object membership such that all pixels in the region belong to the same object. For example, in an image containing a tank, the region containing the tank would have the property that all pixels in the region belonged to the tank.

The space of possible properties that can be used for segmentation form a partial ordering of levels of interpretation of the image.

Much research in image segmentation takes the view that it is a low-level vision concept. Certainly many researchers have attempted to implement segmentation at a low level. Reality seems to be telling us another story.

David Marr gives a graphic example of why segmentation is difficult by showing two overlapping leaves with virtually no discernable edge or texture shift between the two leaves.[3] Identifying the leaves allows us to see the subtle leaf edge that would defeat low-level segmentation. Obviously, using segmentation to aid object recognition is problematic if the segmentation itself requires object recognition. Marr and others suggest that this means that segmentation is an illusion that we have based on introspection of our own capabilities and that it is probably not a low-level phenomenon at all.

When matching manmade maps against aerial-surveillance images, photo interpreters interpret certain patches in the image by combining knowledge of what aerial images look like with knowledge of the manmade maps they are matching against. Medical image segmentation of tissue types of, for example, a heart image might be aided by knowledge from a physiological model of the heart. This is similar to the way that Marr's leaves could be successfully segmented.

In its general form, then, segmentation is not low-level. Practical systems often achieve good segmentations by applying knowledge. We'll refer to regions that have a semantic label as *image content descriptors* and the process of generating such segmentations as *image labeling*. The semantic label is a hypothesis as to that region's content, such as an urban area.

Figure 3 shows an aerial image that has been segmented by several different filters. Combining the evidence from these sources enables fairly accurate segmentations. However, establishing the appropriate structure for the filters and rules for combining the evidence can be problematic. In general, these decisions depend not only on the image's nature and quality but also on the task for which the segmentation is required.

Because image analysis is inherently task-oriented, the analysis requirements will determine the choice of appropriate properties.

Adaptive-labeling approaches. Normally, an image-interpretation program (Figure 4a) combines the results of its filters to yield a set of image content descriptors. The program is developed by selecting an appropriate set of filters for the task, parameterizing the filters, and building the region labeler, all manually.

Such a program is rigid; it cannot be adapted at runtime. Any changes that would cause the program to generate a poor labeling must be determined manually. We seek an approach that lets the system adapt the labeling at runtime, if necessary.

Knowledge-based. One way to achieve greater runtime adaptability is to build a knowledge-based system that contains expert knowledge about

- labeling images;
- the set of filters, which we call the *toolbox*;
- constructing a labeling program; and
- determining the labeling's success.

Figure 4b depicts a system in which an knowledge-based system constructs a labeling program from a toolbox of filters and evaluates the program. In principle, a system like this could be built. However, the AI vision community's experience applying expert systems to vision teaches us that the expert system would soon become unwieldy, and predicting the system's performance would be hard. When new filters are added to the toolbox, the knowledge base will need to expand to accommodate rules for those filters. New expert labeling knowledge will similarly require extending the knowledge base. We would prefer an architecture in which knowledge about the filters was more tightly associated with the filters, and knowledge of image labeling was less ad hoc.

Reflective. Figure 4c shows an adaptive labeler that separates the knowledge of images from the knowledge of filters. Because the filters are reflective, they can evaluate their performance and their applicability to an image-labeling requirement. The knowledge base has been replaced with an image bank in which expert image interpretations have been manually applied. This database should be large enough to permit statistical analysis of

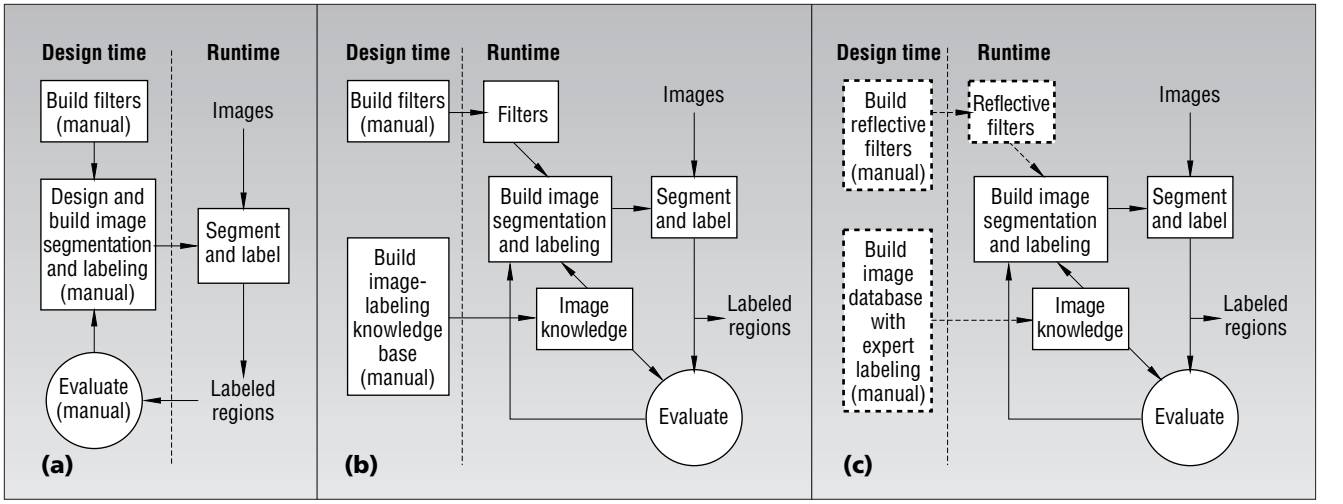

Figure 4. Adaptive vision systems: (a) the standard approach; (b) knowledge-based; (c) reflective.

the likelihood of labelings for the target image types. The box labeled "Image knowledge," which was an expert system in Figure 4b, is now a fixed algorithm for generating labeling programs based on the information in the image bank. Evaluation and labeling are tightly linked because the reflective filters evaluate their own performance.

This reformulation of the architecture removes the need to manually build unwieldy and unpredictable expert systems. It also lets us characterize the runtime system in terms of

- a fixed-image-interpretation program builder that we can analyze so that we are confident about its characteristics, and
- an image labeler structured as a control system. The image labeler evaluates its results and achieves feedback by causing the labeling program to adapt.

The labeling program can be thought of as an algorithm that is parameterized by a set of filters F, a set of parameters P, and a structure S. As the image quality and content changes over a sequence of images, the control system adjusts F, P, and S to produce the best possible image interpretation given the knowledge of images as represented by the image bank and the filters in the toolbox. (We'll further explore the control-system architecture in the next section.)

Not only will such a system generate robust labeling behavior in the face of changing image characteristics, it should enable us to understand its dynamic behavior.

For examples of other architectural approaches to computer vision, see the sidebar.

Persistence and change between labelings. Segmentation, in general, is not static. As imaging satellites orbit the earth, they produce image stripes, regularly passing over the same area. Changes of various forms occur between frames and between passes, including new construction; vehicle movement; and changes in crop cover, terrain type, and weather. Imagery from an airplane provides, on a smaller scale, image stripes that might transition through a number of terrain types.

As a system moves from one context to another, in addition to evidence that the context may be changing from the analysis of the data, there are constraints on the way that the context can change. For example, change in altitude cannot happen spontaneously; the different levels of altitude, which might be represented in, for example, 1,000-foot intervals, form an adjacency diagram that determines which states can be reached from the current one.

By analyzing sequences from the image bank, we can extract not only the adjacency information but also the likelihood of a state transition.

The context consists of a number of markers such as altitude, time of day, and season. Each component might have adjacency constraints. Adjacency diagrams relate contexts. The diagrams can be used in filter selection and calibration.

For example, transitions between adjacent states can help us select between different interpretations of evidence about the context detected by the filter. For example, trees might suddenly change their texture, and the change could be compatible with a sudden change in season from winter to summer or a change from trees to bushes. The adjacency diagram might suggest that the transition to a different season is less likely than a change from trees to bushes.

When a filter can't provide enough (or any) information about the context to support a state change, we can use quality computations to determine when changes should occur by occasionally exploring adjacent states. By using adjacency information, we don't need to search the entire state space as we might have done in a semantics-free approach such as a genetic algorithm. By objectifying (normalizing) quality information so that quality measures from different filters with different models can be compared against the same scale, we can use a hypothesize-and-test approach to search the local adjacency space.

The probability of exploring an adjacent state and the distance along adjacent states that are searched could be a function of quality being reported by the prevailing filter. In this way, when quality starts to drop off, the system starts looking for a more successful filter, model, or topology through a focused local search of the adjacency information. The dynamic behavior of systems such as this will likely be analyzable.

We can obtain the context vocabulary and its structural relationships (adjacency diagram) in several ways:

- *Specification*: We specify the contexts in some language of more primitive tokens. This removes the inductive-learning step necessary in stipulation (see the next bullet) but requires access to subprimitive operators and a language for expressing the specifications. Access to a fixed language of subprimitives reduces the orthogonality of the system's primitive components.
- *Stipulation*: Providing examples, such as in an image corpus, indicates the contexts. The system learns the vocabulary of contexts from these hand-annotated images.
- *Hypothesized*: The contexts are induced directly from representative images.

These three approaches represent differing levels of specification of intent. Specification is programming intent directly in a specification language. Stipulation lets a system procedure optimize an approximation to the intent provided by examples. Hypothesized contexts let the environment drive the adaptation model without any human modeling.

Stipulation, therefore, is interesting as a mechanism for assimilating expert knowledge that cleanly separates intent from the filters involved in its detection and representation.

Hypothesis is interesting because it allows much to be learned from the environment, thereby reducing the extent to which a human developer needs to specify all the details. The hypothesis approach is sympathetic to the "let the environment be its own model" approach, advocated by Rodney Brooks.[4]

These approaches need to be mutually exclusive. An adaptive system might use each of these three mechanisms in varying degrees.

Image analysis as control

Considering an image-processing program to be a closed-loop control system or a collection of such systems is a natural approach to designing a system that performs consistently, even when the environment and perhaps the imaging platform changes. The closed-loop structure is already apparent in Figure 4c.

A control system is an interconnection of components forming a system configuration that will provide a desired response. The foundation for analysis of a conventional control system comes from linear system theory, which assumes a cause-and-effect relationship for the system's components. A closed-loop control system compares feedback from the output signal with the desired input.

Linear system theory is of little help when the components are complex software processes, but we'd like to preserve the ability to

Architectures for computer vision

Early in the development of computer vision, researchers realized that a single sequential thread of processing was inadequate. This was clear even in early experiments aimed at developing printed-character-recognition programs: the recognition step depended on perfect segmentation and enhancement of the printed characters; yet these two processes could benefit from each other's intermediate results and those of the recognition step.

The fundamental reason a single sequential thread does not work well is that such a program resembles a house of cards: failure of the early processes inevitably foils later processes. Because the early processes proved to be (and still are) difficult (if not impossible) to make absolutely reliable, early programs only worked in carefully contrived situations.

It was, and remains, tempting to imagine that simply making the arrows that indicate the flow of control between two processes point in both directions will somehow advance understanding. This "insight" has been the basis for a great deal of work in computer vision, leading to systems that are admirably baroque but mostly do not work, or, if they do work, are quite sensitive to changes to their environment.

One popular variant of this theme was the *blackboard* architecture.[1] In this architecture, independent processes "interact" by reading the intermediate results of other processes (about which they have limited understanding) from a single central data structure called a blackboard, and writing their own results onto the blackboard.

Although the semantics of the blackboard knowledge sources appears simple, it is not represented explicitly. Also, the interactions of the parts, as well the data structures that trigger their interactions, are manually designed. Building and debugging these systems can be every bit as hard as developing a complex tangled web of code. Similar comments apply to other complex software architectures developed for vision over the last 20 years, such as Riseman's vision system.[2] Complex architectures provide the illusion of power, but systems that defy analysis are too dangerous for use in critical applications.

Another idea, borrowed from control theory, is to make explicit a task's constraints and then attempt to maximize agreement with the particular image data while mostly satisfying the constraints. This framework is powerful, but it's limited to representing knowledge as constraints. Further, optimization of nonlinear sets of constraints encourages programs to become trapped in local minima as they search for the global minimum. This, in turn, encourages algorithms that attempt to escape from local minima, such as simulated annealing,[3] genetic algorithms, and graduated nonconvexity.[4]

Recently, considerable interest has focused on perceptual psychology in parallel distributed processes[5,6] and on neural networks, which are computational models of brain operation.[7] However, to date, neither PDP nor neural networks have had much impact on computer vision. One reason is the need for normalization, which is often hard to attain. Mostly, however, the problem seems to be that current neural techniques work from impoverished representations of knowledge. This problem might be solvable.

In light of the limited success of these earlier experiments with architecture, why should we be optimistic that reflection can make a positive contribution? The reflective approach's principal value is that it enables us to represent cleanly and explicitly the architectural framework. Approaches such as neural networks and blackboard systems introduce knowledge at the expense of clarity. Reflection lets us expose the system's semantics so that we can reason about its behavior.

References

1. L.D. Erman et al., "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, Vol. 12, No. 2, June 1980, pp. 213–253.
2. A. Hansen, E. Riseman, and R. Belknap, *The Information Fusion Problem: Forming Token Aggregations across Multiple Representations*, Tech. Report COINS TR87-48, Computer and Information Science Dept., Univ. of Massachusetts, Amherst, Mass., 1987.
3. D.W. Murray, A. Kashko, and H. Buxton, "A Parallel Approach to the Picture Restoration Algorithm of Geman and Geman on an SIMD Machine," *Image and Vision Computing*, Vol. 4, No. 3, Mar. 1986, pp. 133–142.
4. A. Blake and A. Zisserman, *Visual Reconstruction*, MIT Press, Cambridge, Mass., 1987.
5. J.L. McClelland and D.E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, 1986.
6. J.L. McClelland and D.E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2: Psychological and Biological Models*, MIT Press, 1986.
7. C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford Univ. Press, Oxford, UK, 1996.

analyze the resulting software systems to understand the system's expected behavior in terms of robustness and stability.

Robustness is a qualitative attribute related to a system's sensitivity. A robust system's operation is insensitive to parameter variations. Robustness can be characterized by a set of performance indices to system parameters or as the sensitivity of the transfer function.

Stability is defined in terms of a system's response to its input. In a stable system, the response diminishes over time. In an unstable system, the response grows over time. In a marginally stable system, the response doesn't diminish over time but is bounded.

We wish to build systems that are robust and that are stable in that adjustments to the system (changes to the component filters, structure, and parameterization) converge over time.

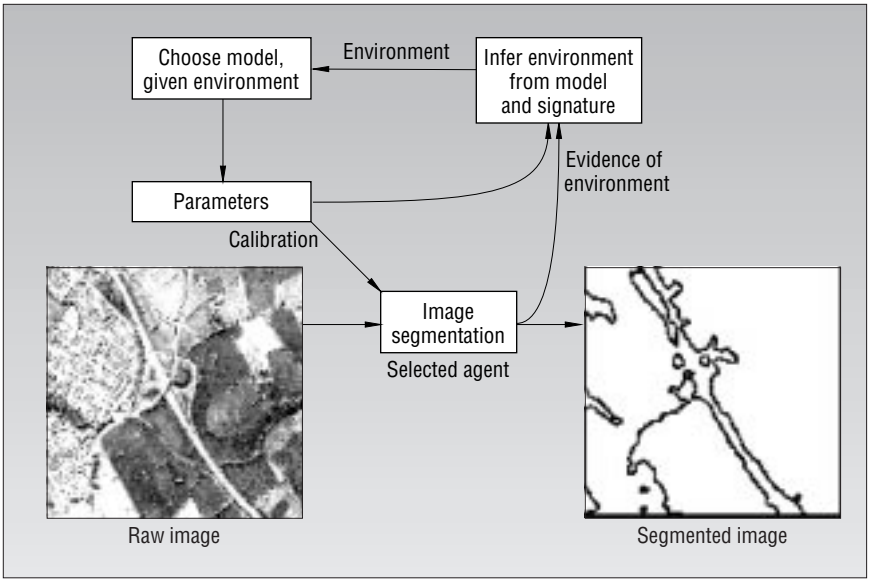

Figure 5. Closed-loop image segmentation with calibration.

Finding a basis for analysis. In an aerial-surveillance program, the image's regions will be labeled as being of specific terrain types—such as "dense forest." The program that performs the segmentation might consist of a collection of filters (algorithms) such as texture analysis, image segmentation, and region labeling. During this discussion, we will not consider specific image-labeling algorithms.

In general, an image-labeling algorithm takes a 2D array of pixels in image space and produces a set of image-content descriptors. These descriptors designate areas in the imaged landscape that give rise to homogeneous (in some measure) regions in the image. The set of parameters that govern the algorithm's behavior and every aspect of image labeling is the model for the algorithm.

In most cases, the filter that produced an image-content descriptor will be able to supply quality information that is subjective to the algorithm-model combination that produced it.

If model space was a real line such that each point along the line was a valid model, if we had a function delta that defined a metric space by telling us for any result how far it was from the ideal result, and if continuous changes in model space caused continuous changes in the delta, we could apply standard control-theory techniques to the filter's calibration. Root locus analysis would let us say certain things about the system's stability. In practice, models are not obviously related to the real line, and the function delta is unlikely to be defined.

The best model for the image labeling is a function of the environment. For example, changes in altitude will affect the expected texture of, say, dense forest.

Aspects of the environment might be detectable. Textures discovered at different altitudes might suggest environmental characteristics. If we have a set of environmental markers, we can seek support for these markers from the properties (such as textures) found during the data analysis.

The likelihood of environmental markers given a property (such as a texture) can be extracted straightforwardly from the image bank, which lets us hypothesize environment marker transitions.

Figure 5 shows how the environment can be inferred from evidence produced by the labeling program.

Given hypotheses for what the imaged environment is, we can use them to select the best model for the algorithm. This can then be generalized to include filter and model selection so that given an environment, we can select appropriate pairs of < *model*, *filter* >. This allows the selection and calibration of filters to occur while the incoming aerial image data is being processed.

When a filter or model changes, a new set of evidence is generated to support the environment. Combining this new evidence with the old will generate a more precise estimate of the environment that might cause the filter or model to change again. For a stable system, this iterative process will cause the environment estimate to converge. The model and filter selection will also converge.

From this simple sketch, we can see that a filter's semantics includes the algorithms involved in calibration, detection, and selection of a set of environment markers. Using a reflective approach, the filters can support a protocol that enables a generic algorithm to ask the filter for its set of environment markers and for evidence that supports those markers, given an image.

A simple elaboration of the approach shown in Figure 5 will let us choose topologies in addition to models and filters. Also, the filters could consider the consumer/producer requirements of interconnected filters when selecting their environment marker sets. By showing that the accumulating evidence for an environmental state converges over time, we can show that the model selection also converges and thus is stable.

In object-oriented languages that support method combination (such as the Common Lisp Object System), a program-construction algorithm—the method-combination algorithm—uses a method's signature to reason about that method's relationship to a specification in the form of a generic function template. Through this process, the algorithm produces the correctly interwoven combined method. A self-adaptive program combines components (methods, agents, filters, or whatever) using an expanded method signature that allows the component's applicability to change based on runtime events and on the static descriptions of the components. The filter developer therefore must be able to write code that can interrogate the process that is trying to use the filter and that can determine dynamically that process's suitability based on the environment. The reflective protocol lets the filter developer extend the semantics of algorithms that support training, selection, calibration, impedence matching, and topology manipulation.

Analysis and correction. Designing closed-loop control systems involves the ability to measure and compare outputs and to apply a

corrective force to the system. We now consider analogs of these abilities for image-analysis programs.

Measuring and comparing outputs. Closed-loop control systems measure the output (and sometimes the input) to detect deviations from a desired signal and to calculate the force necessary to correct the deviation. For example, in an amplifier stage, feedback can be used to hold the amplifier's gain to a preset value.

Applying this approach to image-analysis filters is less straightforward because the nature of the filter's output is different from that of its input. Also, the filters are complex software components that cannot conveniently represent their transfer functions.

For a filter to measure its deviation from its expected behavior, it must have a model of that behavior. Generally speaking, a filter will attempt some kind of interpretation of the input-image features. Sometimes this interpretation will be low-level; other times it will be high-level, such as recognition of complex structures in an image. Whatever the case, the filter should be able to detect problems that it encounters during identification. For example, a filter that attempts to identify settlements in a desert might have difficulty when it encounters a forest. The forest's texture might not fit well with either a settlement or sand dunes. Failures to deal adequately with the input signal will generate a discrepancy signal—a difficulty, failure, or low-confidence interpretation. The discrepancy signal can be quite specific about the problem's nature; for example, it might exactly describe the region containing the trees that it is unable to confidently identify.

Applying the corrective force. Three ways of responding to discrepancy signals are of particular interest because they involve reasoning about the computational nature of the image-analysis program itself:

- Dynamically change filters as needs change by replacing old filters with better ones as they become available. This is the filter-selection problem.
- Autocalibrate filters to work optimally with the data they are consuming. This is the filter-refinement or autocalibration problem.
- Let filters modify their input signal expectations and the form of their output to bring interconnected filters into closer alignment with their respective needs. This is the inter-filter-communication problem, sometimes called the impedance-matching problem.

AN ADAPTIVE APPROACH TO AERIAL-surveillance image analysis has these related benefits:

- The image-analysis process can smoothly adapt to changes in image quality that would otherwise lead to abrupt changes in interpretation quality.
- Appropriate resources can be selected for an image's needs in a way that allows the efficient use of resources. For example, filters could be implemented in hardware using FPGAs. The adaptive mechanism provides a natural way of reprogramming FPGAs as needed to deal with changing interpretation needs and image conditions.
- The system requires less tinkering to make it work.
- Reflection allows the knowledge of the filters, task, and imaging environment to guide the automatic integration of filter results. (Normally, this integration is manual and inadequate.)
- The image bank provides a convenient mechanism for converting expert (photo interpreter) image-analysis knowledge into a form usable for control and adaptation.

However, this approach has two problems:

- Filter implementation requires significant additional effort. In particular, when designing filters for use in image-analysis applications, we must provide for the filter to monitor its own performance so that it can report divergence from its intended behavior.
- Stability analysis is typically part of control system design, but generalizing stability analysis for image analysis might be difficult. Part of the solution might be better tools for developing, debugging, and testing adaptive systems.

The ideas in this article have been implemented in the form of a reflective agent architecture. Two aerial image corpora have been produced with manually annotated ground truth. One corpus is based on 105 512×512 multispectral images from the European SPOT satellite, the other is based on 40 2000×2000 pixel Massachusetts GIS grayscale images taken from a plane at 10,000 feet. By treating the manual annotations as a specification of desired performance of the image segmentation and labeling process, we have been able to use that specification to automatically compose an image parser. We have also allowed the parser process to change the selection of filters used in providing supporting evidence for the segmentation and labeling of the image to produce the best parse of the image from the standpoint of the specification. For more technical details of our project, visit our Web page on adaptive aerial-image analysis, *www.robots.ox.ac.uk/~pr/aaia.html*. ❒

Acknowledgments

The Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, sponsored this research under agreement F30602-98-0056. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions in this article are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Research Laboratory, or the US Government.

References

1. B.C. Smith, *Reflection and Semantics in a Procedural Language*, Tech. Report 272, MIT Laboratory for Computer Science, Massachusetts Inst. of Technology, Cambridge, Mass., 1982.
2. Z.-Y. Xie and J.M. Brady, "Texture Segmentation Using Local Energy in Wavelet Scale Space," *ECCV J.*, Vol. A, 1996, pp. 304–313.
3. D. Marr, "Early Processing of Visual Information," *Philsophical Trans. Royal Soc. London B*, Vol. 275, 1976, pp. 483–524.
4. R.A. Brooks, "Intelligence without Representation," *Artificial Intelligence*, Vol. 47, Jan. 1987, pp. 139–159.

Paul Robertson is chief technical officer and cofounder of Dynamic Object Language Labs Inc.—a company dedicated to advanced software and AI. He has an interest in computational reflection, computer vision, advanced languages and architectures for building intelligent systems. He holds a BA in computer science, and he is a PhD candidate at the University of Oxford. Contact him at pr@robots.ox.ac.uk.

J. Michael Brady is a professor of information engineering and the founder of the Robotics Laboratory and the Medical Vision Laboratory at the University of Oxford. He received his PhD in mathematics.